

1. Einleitung
 2. Grundlagen
 - 2.1. BPMN
 - 2.2. Repository Mining
 3. Mining von BPMN-Prozessen auf GitHub
 - 3.1. Ansätze
 - 3.1.1. GHTorrent
 - 3.1.2. Klonen
 - 3.1.3. Github API
 - 3.2. GitHubBpmnCrawler Implementierung
 - 3.3. Ergebnisse
 4. Analyse gefundener BPMN-Prozesse
 - 4.1. Allgemeine Statistiken: Programmiersprache, Dateitypen, Alter, etc.
 - 4.2. BPMNSpector
 5. Zusammenfassung und Ausblick
- Referenzen
- Anhang (optional)

1 Einleitung

2 Grundlagen

Das folgende Kapitel gibt einen kurzen Überblick über BPMN-Diagramme und das Mining Software Repositories, als Forschungsgebiet.

2.1 BPMN

Die Business Process Model and Notation (BPMN, deutsch Geschäftsprozessmodell und -notation) ist eine Spezifikation, die eine grafische Notation zum Ausdrücken von Geschäftsprozessen in einem Geschäftsprozessdiagramm (englisch Business Process Diagram, BPD) bereitstellt. Sie ist ein öffentlicher Standard, der von OMG und BPMI verwaltet wird. Das gesamte Dokument, das BPMN ausführlich beschreibt, umfasst mehr als fünf hundert Seiten und kann von der offiziellen Internetseite OMG kostenlos heruntergeladen werden. Siehe: [2].

Das Hauptziel von BPMN, wie in der Spezifikation steht, ist es, eine Notation bereitzustellen, die für alle Geschäftsbenutzer leicht verständlich ist, von den Business-Analysten, die die ersten Entwürfe der Prozesse erstellen, bis zu den technischen Entwicklern, die für die Implementierung der Technologie verantwortlich sind, die diese Prozesse durchführt, und schließlich auf die Geschäftsleute, die diese Prozesse verwalten und überwachen. Somit schafft BPMN eine standardisierte Brücke für die Lücke zwischen Geschäftsprozessdesign und Prozessimplementierung [18].

Grundsätzlich basiert BPMN auf der Überarbeitung anderer Notationen und Methoden, insbesondere des Aktivitätsdiagramms der Unified Modeling Language (UML), des UML-EDOC-Geschäftsprozesses, IDEF, ebXML BPSS, des Aktivitäts-Entscheidungsflussdiagramms, RosettaNet, LOVeM und Event Driven Process Chains [16]. Die aktuelle Version von BPMN (2.0.2) definiert unter anderem das XML-basierte Format für das Speichern und die Übertragung von BPMN-Diagrammen.

Insgesamt werden in der Spezifikation zur BPMN vier Arten von BPD definiert:

- Prozessdiagramm (Process Diagram)
- Kollaborationsdiagramm (Collaboration Diagram)
- Choreographie-Diagramm (Choreography Diagram)
- Konversationsdiagramm (Conversation Diagram)[18].

Für die Bildung von BPD stellt Spezifikation zur BPMN eine Menge von Elementen sowie Informationen für ihre Bedeutung und Regeln, wie sie kombiniert wer-

den können, bereit. Auf solche Weise werden Syntax und Semantik von BPMN-Diagrammen geregelt [10]. Jedes Element gehört einer der fünf Kernelementkategorien [2]. Diese sind:

1. **Flow Objects** (Knoten)
2. **Data** (Daten)
3. **Connecting Objects** (Verbindende Objekte)
4. **Swimlanes** (Schwimmbahnen, die Teilnehmer darstellen)
5. **Artifacts** (Artefakte)

In der Tabelle 2.1 werden die wichtigsten Elemente von BPMN-Diagrammen dargestellt.

Tabelle 2.1: Grundlegende Modellierungselemente von BPMN-Diagrammen[1], [10]

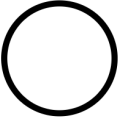

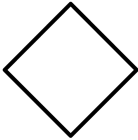
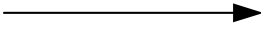
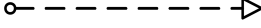
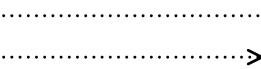

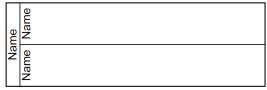
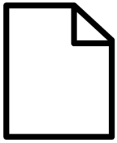


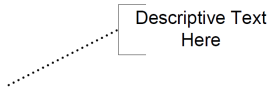
Element	Beschreibung	Grafische Notation
Event (Ereignis)	Ereignis markieren Zeitpunkte. Es gibt drei Arten von Ereignissen, je nachdem, wann sie den Fluss beeinflussen: Start, Zwischenzeit und Ende.	
Activity (Aktivität)	Eine Aktivität ist ein Oberbegriff für die Arbeit, die ein Unternehmen in einem Prozess ausführt. Eine Aktivität kann atomar oder nicht atomar sein.	
Gateway (Entscheidungspunkt)	Ein Entscheidungspunkt wird verwendet, um die Divergenz und Konvergenz von Sequenzflüssen zu steuern. Es gibt drei Arten von Entscheidungspunkten: XOR-Gateway, auch exklusives Gateway lässt genau einen Prozesspfad zu; UND-Gateway, paralleles Gateway (alle Pfade müssen durchschritten werden); OR-Gateway, inklusives Gateway (es muss mindestens ein Pfad gewählt werden).	
Sequence Flow (Sequenzfluss)	Ein Sequenzfluss wird verwendet, um die Reihenfolge anzuzeigen, in der Aktivitäten ausgeführt werden.	
Message Flow (Nachrichtenfluss)	Ein Nachrichtenfluss wird verwendet, um den Nachrichtenfluss zwischen zwei Teilnehmern anzuzeigen, die bereit sind, sie zu senden und zu empfangen.	
Association (Assoziation)	Eine Assoziation wird verwendet, um Informationen und Artefakte mit BPMN-Grafikelementen zu verknüpfen.	

Tabelle 2.1: Grundlegende Modellierungselemente von BPMN-Diagrammen

Element	Beschreibung	Grafische Notation
Pool (Schwimmbecken)	Ein Pool ist die grafische Darstellung eines Teilnehmers an einer Kollaboration. Es fungiert auch als Swimlane und als grafischer Container, um eine Gruppe von Aktivitäten von anderen Pools zu trennen. Ein Pool kann einen Prozess referenzieren, muss aber nicht. D.h. ein Pool kann auch als eine Blackbox sein.	
Lane (Spur)	Eine Spur ist eine Unterpartition innerhalb eines Prozesses, manchmal innerhalb eines Pools, und erstreckt sich über die gesamte Länge des Prozesses, entweder vertikal oder horizontal.	
Data Object (Datenobjekt)	Datenobjekte bieten Informationen dazu, welche Aktivitäten ausgeführt werden müssen und / oder was sie produzieren. Datenobjekte können ein einzelnes Objekt oder eine Sammlung von Objekten darstellen.	
Message (Nachricht)	Eine Nachricht wird verwendet, um den Inhalt einer Kommunikation zwischen zwei Teilnehmern darzustellen.	
Group (Gruppe)	Eine Gruppe ist eine Menge grafischer Elemente, die sich in derselben Kategorie befinden. Gruppen bieten die Möglichkeit, Kategorien von Objekten visuell im Diagramm anzuzeigen.	
Text Annotation (Textanmerkung)	Textanmerkungen sind ein Mechanismus für einen Modellierer, um dem Leser eines BPMN-Diagramms zusätzliche Textinformationen bereitzustellen.	

Wir möchten uns in der Abbildung 2.1 ein einfaches BPMN-Prozessdiagramm anschauen.

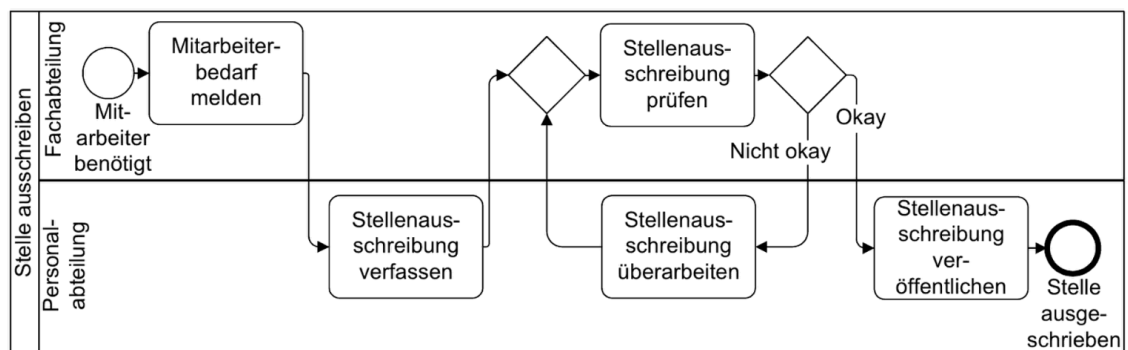


Abbildung 2.1: Ein einfaches BPMN-Prozessdiagramm. Übernommen aus [9].

Der Prozess „Stelle ausschreiben“, an dem Fachabteilung und Personalabteilung beteiligt sind, beginnt wenn ein neuer Mitarbeiter benötigt wird. Wenn das der Fall ist, meldet die Fachabteilung den Mitarbeiterbedarf. Nachdem das gemacht wird, verfasst die Personalabteilung eine Stellenausschreibung, die von Fachabteilung überprüft wird. Falls sie nicht okay ist, wird sie von Personalabteilung überarbeitet und wieder von Fachabteilung überprüft, wobei sich dieser Vorgang mehrmals wiederholen kann. Wenn Stellenausschreibung okay ist, wird sie von Personalabteilung veröffentlicht. Somit ist der Prozess „Stelle ausschreiben“ zu Ende [9].

Jetzt versuchen wir die Abbildung 2.1 nach Mal zu beschreiben und dabei die dargestellte Elemente zu nennen. Insgesamt besteht dieses Diagramm aus folgenden Elementen:

- Pool
- Lane
- Start Event
- Activity
- Sequence Flow
- XOR-Gateway
- End Event

Das gesamte Prozess befindet sich innerhalb des Pools und hat den Namen „Stelle ausschreiben“. Das Pool ist in zwei Bahnen (eng. „Lanes“) geteilt. Die Teilung in Bahnen macht es deutlich, welche Aktivitäten von Fachabteilung und welche von Personalabteilung gemacht werden. Der Prozess startet mit Start Event. Die vom Start Event ausgehende Kante (Sequence Flow) zeigt, welches Element als Nächstes durchlaufen wird. So kommen wir zur Aktivität „Mitarbeiterbedarf melden“, der die Aktivität „Stellenausschreibung verfassen“ folgt. Nach letzter passieren wir einen XOR-Gateway, der nur eine ausgehende Kante (Sequence Flow) hat. Somit erreichen wir die Aktivität „Stellenausschreibung prüfen“. Dieser Aktivität folgt das zweite XOR-Gateway mit zwei ausgehenden Kanten. Jetzt kann es wegen der Exklusivität der XOR-Gateway nur in eine Richtung weiter gehen. Die Richtung der Sequenz wird durch die Bedingungen „Okay“ oder „Nicht Okay“ bestimmt. Somit kann es wieder zur Aktivität „Stellenausschreibung prüfen“ und der beschriebenen Vorgang wiederholt sich, oder zur Aktivität „Stellenausschreibung veröffentlichen“. Nach dieser Aktivität wird das letzte Element, nämlich End Event, erreicht, was das Ende des Prozesses bedeutet.

2.2 Mining Software Repositories

Das Mining Software Repositories (MSR) -Forschungsgebiet analysiert und verbindet die umfangreichen, in den Softwareprojekten verfügbaren Daten, um interessante und umsetzbare Informationen zu Softwaresystemen und -projekten zu finden [13].

Beispiele für Software-Repositories sind:

- **Historische Repositories** wie Quellcodeverwaltungs-Repositories, Fehler-Repositories und archivierte Kommunikationen zeichnen verschiedene Informationen über die Entwicklung und den Fortschritt eines Projekts auf [13].
- **Laufzeit-Repositories** wie Bereitstellungsprotokolle enthalten Informationen zur Ausführung und Verwendung einer Anwendung an einer oder mehreren Bereitstellungsstandorten [13].
- **Code-Repositories** wie Git-Versionsverwaltungssysteme GitHub und GitLab oder SourceForge. Solche Hosting-Plattformen enthalten den Quellcode verschiedener Anwendungen, die von mehreren Entwicklern entwickelt wurden.

Programmierer erstellen und sammeln während der Softwareentwicklung viele Daten, die alle automatisch abgerufen und analysiert werden können. Wie zum Beispiel:

- Der Quellcode des Programms. Dies ist die wichtigste Eingabe für die mögliche Analyse.
- Durch die während der Ausführung der Software gesammelte Daten können Profile erhalten werden, die mitteilen, welche Teile der Software häufig verwendet werden und welche nicht.
- Den Produkten liegen möglicherweise zusätzliche Dokumentationen bei, beispielsweise Konstruktionsdokumente oder Anforderungsdokumente. Diese können auch wichtige Funktionen enthalten, die erklären, warum Code so aussieht, wie er aussieht.
- Die resultierende Software kann statisch analysiert werden und bietet Funktionen wie Komplexitätsmetriken oder Abhängigkeiten.
- Versionsarchive zeichnen die am Produkt vorgenommenen Änderungen auf, einschließlich wer, wann, wo und warum. Versionsarchive können viel über die Historie eines Projekts aussagen, wenn die gespeicherten Änderungen alle logisch getrennt sind und die gespeicherten Rationen systematisch und konsistent verwendet werden [15].

Der Algorithmus für das MSR kann wie folgt formuliert werden:

1. **Ziel des MSR bestimmen**, d.h. welche Daten sollen gesammelt werden. Diese Daten werden später für die Analyse verwendet.
2. **Datenabruf**. Um die ständige Verfügbarkeit der gezielten Daten sicherzustellen und einen schnellen Zugriff auf sie zu ermöglichen, sollten lokale Kopien erstellt werden.
3. **Datenkonvertierung** (optional). Data Mining erfordert, dass die Daten nicht nur heruntergeladen und verarbeitet werden, sondern auch, dass man viele ihrer Merkmale im Voraus versteht. Daher sollten die Daten den Voraussetzungen und Annahmen entsprechen, die vorher getroffen wurden.

4. **Datenextraktion.** Nachdem die Daten auf lokaler Festplatte in einer geeigneten Form gesichert wurden, können Sie verarbeitet werden. Die Verarbeitung umfasst das Extrahieren, Filtern und Speichern der Inhalte der Ressourcen in einem dauerhaften und für den Menschen lesbaren Format. Zum Beispiel die gefilterte Informationen können in einer relationalen Datenbankmanagementsystem gespeichert werden.
5. **Datenanalyse.** In diesem Schritt wird die gezielte Analyse der gespeicherten Daten durchgeführt [15].

3 Mining von BPMN-Prozessen auf GitHub

3.1 Ansätze

In Vorbereitungsphase unserer Forschung wurden verschiedene Ansätze für die Suche nach GitHub-Projekte, die BPMN-Diagramme enthalten, untersucht und ausprobiert. Grundlage dafür lieferten uns Publikationen, in denen es um die Suche nach UML-Diagramme enthaltende GitHub-Projekte ging. Siehe: [17], [14], [21] und [20].

3.1.1 GHTorrent

Entsprechend der offiziellen Quellen von GitHub verfügt diese Plattform weltweit über mehr als 31 Million registrierter Benutzer und Mehr als 100 Millionen Projekte [5], wobei fast ein Drittel von ihnen innerhalb des letzten Jahres erzeugt wurden [8]. Diese Zahlen zeigen, dass GitHub sehr dynamische Hosting Plattform ist und einen sehr großen Bestand von Daten enthält. Um diese Daten analysieren zu können, wird eine statische, zuverlässige und aktualisierte Quelle dieser Plattform gebraucht [17]. Als solche Quelle wurde von uns das GHTorrent-Projekt ausgewählt [4].

GHTorrent-Projekt wurde entwickelt, wie seine Gründer angeben, um das Studium von GitHub zu erleichtern. Dieses Projekt bietet einen skalierbaren, abfragbaren Offline-Spiegel der GitHub-Daten, die von GitHub-REST-API angeboten werden [12]. GHTorrent aktualisiert seine Daten jeden Monat. Die können sowohl heruntergeladen als auch ohne herunterzuladen abgefragt werden. Herunterladen kann man sie als MySQL-Dump (Menge von csv-Dateien, jede von denen einer MySQL-Datenbanktabelle entspricht) oder als MongoDB-Dump. Das Abfragen ist möglich sowohl durch eine GHTorrent-Online-Abfrageoberfläche [12] oder mithilfe von Google BigQuery [6], das einen aktuellen Import des neuesten MySQL-Dumps von GHTorrent enthält [3].

Nachdem die MySQL-Dump als tar-Archiv mit 23 unterschiedlichen csv-Dateien und MySQL-Datenbankschema heruntergeladen wurde, haben wir Dateien „users.csv“, die Informationen über GitHub-Benutzer enthält, und „projects.csv“ mit Informationen über GitHub-Repositorys in gleichnamige Datenbanktabellen importiert. Siehe Abbildung 3.1. Diese Daten haben es möglich gemacht, eine Liste von nicht gelöschten und nicht abgezwigten GitHub-Repositorys zu bilden.

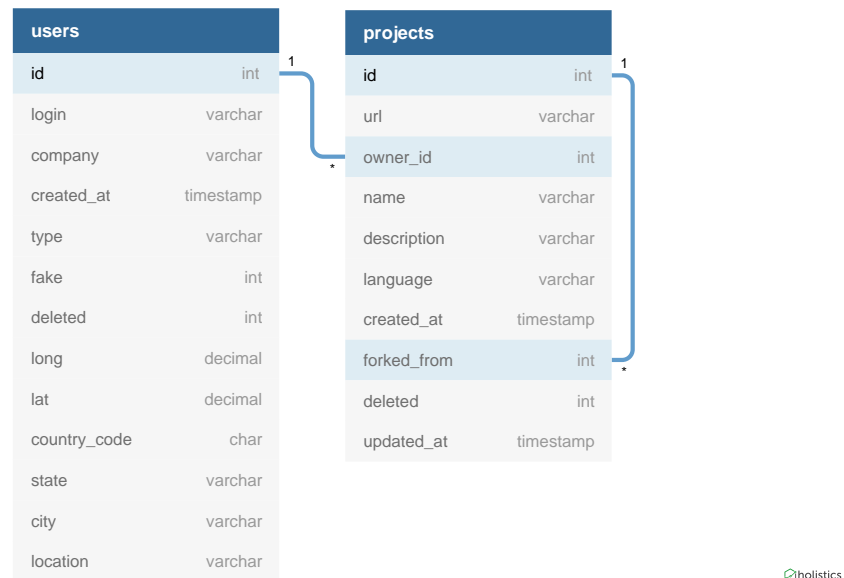


Abbildung 3.1: MySQL-Datenbankschema von Tabellen „users“ und „projects“

GHTorrent stellt keine Informationen zu Dateien aus einem bestimmten GitHub-Projekt zur Verfügung [17]. Es ist daher erforderlich, diese Informationen entweder mit `git clone` oder über GitHub API von GitHub abzurufen.

3.1.2 Klonen

Da GitHub die verteilte Versionskontrolle Git als Standardmechanismus für die Verwaltung von Softwaresystemen verwendet, war es möglich mithilfe des `git clone` Befehls, die GitHub-Projekte herunterzuladen und dann nach BPMN-Diagrammen zu durchsuchen.

Es wurde ein Python Programm für die Suche nach Projekten mit BPMN-Diagrammen geschrieben. In diesem Programm wurde in einer Schleife über die Liste mit Namen der GitHub-Projekte iteriert und bei jeder Iteration mit `git clone` ein Projekt heruntergeladen und durchsucht.

Trotz der richtigen Ergebnissen, die dieses Verfahren lieferte, war es zu langsam, wenn es um große Repositorys ging. (Dieses Problem wurde in [11] beschrieben). Das Problem besteht darin, dass beim Klonen eines Repositorys nicht nur Dateien, sondern auch ganze Geschichte des Projekts heruntergeladen wird. Bei großen Projekten werden bis zu mehreren GB Daten heruntergeladen, was die Zeit kostet. Deshalb stellte sich dieses Verfahren für unsere Forschung als ineffektiv heraus.

3.1.3 Github API

GitHub bietet eine Anwendungsprogrammierschnittstelle (englisch: Application Programming Interface, kurz API) für den Zugriff auf Daten in öffentlichen Repositorys. Der gesamte API-Zugriff erfolgt über HTTPS und kann über `https://api.github.com` aufgerufen werden. Alle Daten werden als JSON ge-

sendet und empfangen [7].

Die GitHub-API definiert eine Beschränkung von 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer und 60 Anfragen pro Stunde für nicht authentifizierte. Nicht authentifizierte Anfragen werden der IP-Adresse zugeordnet und nicht dem Benutzer, der Anfragen stellt [7].

3.1.3.1 Github API Search

Die Such-API ermöglicht die Suche nach dem gewünschten Artikel. Es können, beispielsweise, spezifische Daten (z. B. Code, Sterne, Commits oder Mitwirkende) in einem oder in mehreren Projekten gesucht werden. Allerdings gibt es folgende Beschränkungen bei der Such-API:

- Es können höchstens 30 authentifizierte oder 10 nicht authentifizierte Anfragen pro Minute gemacht werden.
- Die Antwort für eine Anfrage kann höchstens 1000 Resultate haben.
- Es wird nur der Standardzweig (`default branch`) bei der Suche berücksichtigt. In den meisten Fällen ist dies der Hauptzweig (`master branch`).
- Nur Dateien, die kleiner als 384 KB sind, können durchsucht werden [7].

Es wurde wiederum ein Python-Programm geschrieben, in dem in einer Schleife über die Liste mit Namen der GitHub-Projekte iteriert wurde und bei jeder Iteration eine Anfrage über GitHub API gesendet wurde. Für die Suche nach Schlüsselwörter `KeyString` wurde für jeden GitHub-Benutzer `ExampleUser` und Repository `ExampleRepository` folgende Anfrage gesendet: `https://api.github.com/search/code?q=KeyString+in:file+user:ExampleUser+repo:ExampleRepository`.

Dieser Ansatz hatte wichtige Vorteile im Vergleich zum oben beschriebenen **git clone**. Erstens es lieferte ein JSON-Objekt mit Pfaden zu den Dateien, die die gesuchte Schlüsselwörter enthalten. Somit hatten wir Informationen, welche Dateien innerhalb jeweiliges Projektes zu extrahieren sind. Und zweitens, es war deutlich schneller.

Allerdings gab es auch Nachteile im Vergleich zum Klonen. Zum einen konnten Dateien größer als 384 KB nicht durchsucht werden. Und zum anderen konnten nur 1000 Resultaten für ein Projekt ermittelt werden. Folglich konnten bei sehr großen Projekten nicht alle Ergebnisse ermittelt werden.

Insgesamt konnten mithilfe von Such-API 1800 Repositories pro Stunde durchsucht werden.

3.1.3.2 Durchsuchen der Baumstruktur

Es gab noch ein Verfahren, das das Benutzen des GitHub API möglich machte. Deswegen Besonderheit ist es, dass nicht der Inhalt einer Datei in entsprechendem Projekt nach Schlüsselwörter durchsucht wird. Sondern es wird zuerst die Projektstruktur

tur als Baum mit Dateinamen und Ordnern bestimmt und nur in Dateinamen einschließlich der Dateiendung nach Schlüsselwörter gesucht [17].

Dieses Verfahren besteht aus drei Schritten:

- Ermitteln des letzten `commit`, das dem Standardzweig eines Repositorys gehört. Dafür wird eine authentifizierte `http`-Anfrage an die GitHub-API mit (`branches/master`) als Parameter gesendet. Falls `master branch` der Standardzweig, also `default branch` ist, enthält Ergebnis der Anfrage das letzte `commit`, was das Ziel war.

Es kann allerdings vorkommen, dass `master branch` nicht der Standardzweig (`default branch`) ist. In diesem Fall wird die zweite Anfrage gesendet, um den Standardzweig zu ermitteln. Es sei Standardzweig `defBr`, dann die dritte Anfrage, die gesendet wird, enthält (`branches/defBr`) als Parameter. So wird auch hier das letzte `commit` ermittelt.

- Senden einer authentifizierten `http`-Anfrage, um die Baumstruktur des jeweiligen Repositorys anhand des letzten `commit` zu erhalten.
- Suchen in „Blätter“ des Baums, die Dateinamen jeweiliges Projekts sind, nach Schlüsselwörter. Falls gefunden, Speichern des Dateipfads zu einer Datenbank.

Dieses Verfahren enthält in sich Vorteile des Klonens und Github API Search. D. h. es ist schnell und es werden alle Repositorys und ihre Dateinamen unabhängig von ihren Größen durchsucht.

Wegen der Beschränkung des GitHub API, können bei diesem Ansatz höchstens 5000 authentifizierte `http`-Anfragen gemacht werden. Für jedes Repository werden mindestens zwei Anfrage gestellt: erste fürs Ermitteln des letzten `commit` und zweite fürs Ermitteln der Baumstruktur. Es können aber noch zusätzliche zwei Anfragen gebraucht werden. Es ist dann der Fall, wenn der Hauptzweig nicht `master branch` ist. Aus diesen Gründen können bei diesem Verfahren theoretisch mindestens 1250 und höchstens 2500 GitHub-Projekten pro Stunde durchsucht werden.

Das Benutzen dieses Ansatzes stellte heraus, dass die Mehrheit des GitHub-Projekten `master branch` als Standardzweig hat. Daher war die Anzahl der GitHub-Projekte, die pro Stunde durchsucht werden könnten, nah zur oberen Schranke mit 2500 GitHub-Projekten. Somit war dieser Ansatz auch in diesem Sinne effektiver als Klonen und GitHub API Search.

3.2 „GitHubBpmnCrawler“ Implementierung

Nach der Analyse verschiedener Ansätze haben wir GHTorrent, als statische Quelle von GitHub-Projekten ausgewählt. Diese Ressource bereitete uns Namen aller GitHub Benutzer und öffentlichen Repositorys. Weil sie keine Informationen über Dateien in jeweiligen Repository enthält, wurde GitHub API mit in 3.1.3.2 beschriebenem Ansatz der Durchsuchung der Projektsbaumstruktur genommen.

Die Arbeit an unserem Programm startete am Anfang des Novembers 2018. Es wurde GHTorrent MySQL-Dump von 01.11.2018 heruntergeladen und daraus MySQL-Datenbanktabellen „users“ und „projects“ (Siehe Abbildung 3.1) extrahiert. Diese zwei Tabellen enthalten Daten, die es möglich machen, eine Liste von nicht gelöschten und nicht abgezweigten GitHub-Repositorys zu bilden. Die SQL-Anfrage dafür sieht wie folgt aus:

```
SELECT users.login, projects.name FROM users, projects
WHERE projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL.
```

Das Ergebnis dieser Anfrage enthielt Namen und Logins von 61.632.173 GitHub-Repositorys. Wegen der Beschränkung der GitHub API, die höchstens 5000 authentifizierte http-Anfragen pro Stunde erlaubt, könnte die Durchsuchung dieser GitHub-Repositoryn im bestem Fall (siehe obere Schranke in 3.1.3.2) 1.027 Tage dauern. Weil diese Dauer zu groß wäre, wurde die Entscheidung getroffen, nur 10 Prozent von gesamten Anzahl, d.h. 6.163.217 GitHub-Repositoryn, zu durchsuchen. Diese Anzahl wiederum konnte im besten Fall in 103 Tage durchsucht werden.

Um die Durchsuchung der GitHub-Repositoryn zu beschleunigen, sollte das Programm so entwickelt werden, dass es an einem Rechner mehrfach gestartet werden kann. D.h. es sollte so programmiert werden, dass mehrere Instanzen des Programms mit unterschiedlichen Anmeldedaten des registrierten GitHub API Benutzers initialisiert werden und gleichzeitig unterschiedliche Repositorys durchsuchen können.

3.2.1 Datenbank von „GitHubBpmnCrawler“

Die Abbildung 3.2 zeigt uns die Struktur der Datenbank des „GitHubBpmnCrawler“ Programms, die als SQLite Datenbank implementiert wurde.

Die **users** Tabelle enthält Informationen über GitHub Benutzer und zwar Login (*login*) und Ländercode (*country_code*).

Die Tabelle **projects** beschreibt ein GitHub Repository. Sie enthält Name des Projekts (*name*), Verweis auf Inhaber (*owner_id*) in Tabelle **users**, Attribut *forked_from*, in dem gespeichert wird, aus welchem Repository es abgezweigt wurde, wenn das überhaupt der Fall ist, und Attribut *deleted*, wo steht, ob Repository gelöscht wurde.

Die Inhalte der **users** und **projects** Tabellen wurden aus gleichnamigen Tabellen der MySQL Datenbank, die aus GHTorrent Archiv stammt, importiert. Siehe Abbildung 3.1.

Die Tabelle **to_query_projects** dient als Eingabequelle für das „GitHubBpmnCrawler“ Programm. Sie enthält Login (*login*) und Name (*name*) von 10 Prozent der nicht gelöschten und nicht abgezweigten GitHub-Repositorys. Im Attribut *status* wird gespeichert, ob das Repository durchsucht wurde (Wert 1) oder zu durchsuchen ist (Wert 0). Die Daten wurden in diese Tabelle randomisiert aus **users** und **projects**

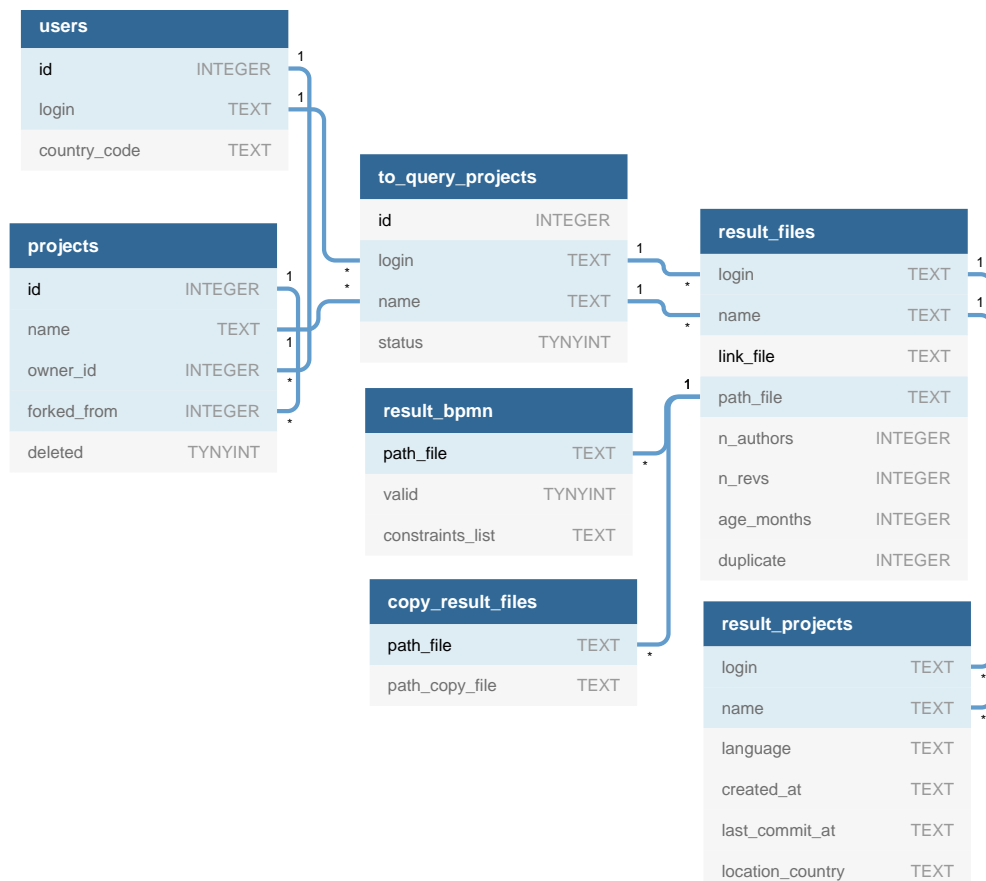


Abbildung 3.2: Datenbankdiagramm

mit folgender Anfrage eingefügt:

```
INSERT INTO to_query_projects (login, name) SELECT
users.login, projects.name FROM users, projects WHERE
projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL ORDER BY RANDOM() LIMIT 6163217.
```

Die Tabelle **result_files** enthält Informationen über Dateien, die das „GitHubBpmnCrawler“ Programm als potentielle BPMN-Diagramme eingestuft hat. Sie enthält folgende Attribute: Login des Inhabers (*login*), Name vom GitHub Repository (*name*), Weblink (*link_file*), Pfad innerhalb des Projektes (*path_file*), Anzahl der Autoren (*n_authors*) und Änderungen (*n_revs*), Alter der Datei in Monaten (*age_months*) und Attribut *duplicate*, wo steht, ob Duplikate der entsprechenden Datei gefunden wurden.

Die Tabelle **result_bpmn** enthält Informationen über gefilterten BPMN-Diagramme im XML-Dateiübertragungsformat. Das Attribut *path_file* referenziert gleichnamiges Attribut aus der **result_files** Tabelle und hat die gleiche Bedeutung. Das *valid* Attribut enthält Information, ob BPMN-Diagramm keine Restriktionen (engl. Constraints) hat. Im Attribut *constraints_list* wird eine Liste der Restriktionen gespeichert, die im BPMN-Diagramm gefunden wurden.

Die Tabelle **copy_result_files** enthält zwei Attribute: *path_file* und *copy_path_file*. Das erste Attribut referenziert gleichnamiges Attribut aus der **result_files** Tabelle und hat die gleiche Bedeutung. In dem zweiten Attribut wird der Pfad gespeichert,

wo sich die Kopie der Datei befindet.

Die Tabelle **result_projects** enthält Informationen über Repositories, wo mindestens ein potentielles BPMN-Diagramm gefunden wurde. Es sind: Login des Inhabers (*login*), Name vom GitHub Repository (*name*), die dominante Programmiersprache des Projekts (*language*), die Zeit, wann Repository erstellt wurde (*created_at*) und letztes Mal geändert wurde (*last_commit_at*). Im Attribut (*location_country*) wird den Namen des Landes gespeichert, aus welchem das Repository stammt.

3.2.2 „GitHubBpmnCrawler“

Die Abbildung 3.3 zeigt uns den gesamten Prozess, der mit dem „GitHubBpmnCrawler“-Programm realisiert wird. Er besteht aus drei Schritten:

- (1) Im ersten Schritt werden potentielle BPMN-Diagramme aus öffentlichen GitHub-Repositories gesammelt.
- (2) Im zweiten Schritt wird aus der Menge der potentiellen BPMN-Diagramme eine Untermenge der BPMN-Diagramme gefiltert.
- (3) Im letzten Schritt werden Statistiken über BPMN-Diagramme gebildet.

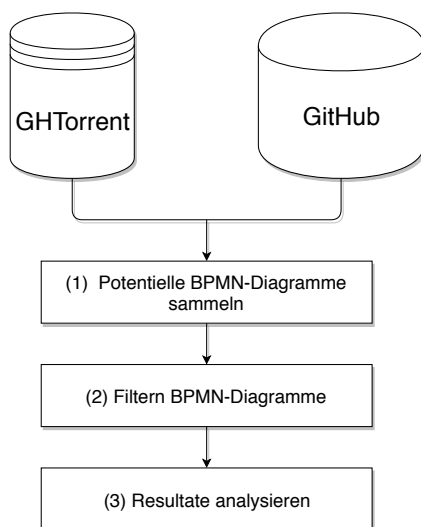


Abbildung 3.3: Der gesamte Prozess des „GitHubBpmnCrawler“-Programms

Das Klassendiagramm 3.4 zeigt die Struktur von „GitHubBpmnCrawler“. Es sind neun Klassen zu sehen. Innerhalb der jeweiligen Klasse gibt es Methodennamen und ihre Rückgabetyps. Methodenparameter und Attribute wurden sowohl aus Platzspargründen, als auch, weil sie in diesem Kontext nicht von besonderen Bedeutung sind, nicht dargestellt. Die Methoden der Klassen *RepositoryCrawler* und *TreeCrawler* haben wir von Gregorio Robles entwickeltem „2016-uml-miner“-Programm [19] übernommen und für unsere Zwecke angepasst.

Die **GitHubApiCrawler** Klasse enthält Methode *run_api_crawler()*, die vollautomatisiert ersten Schritt aus der Abbildung 3.3 abdeckt. Sie bekommt als Parameter zwei Integer Variablen, die das Minimum und das Maximum *id* für Repositorien in

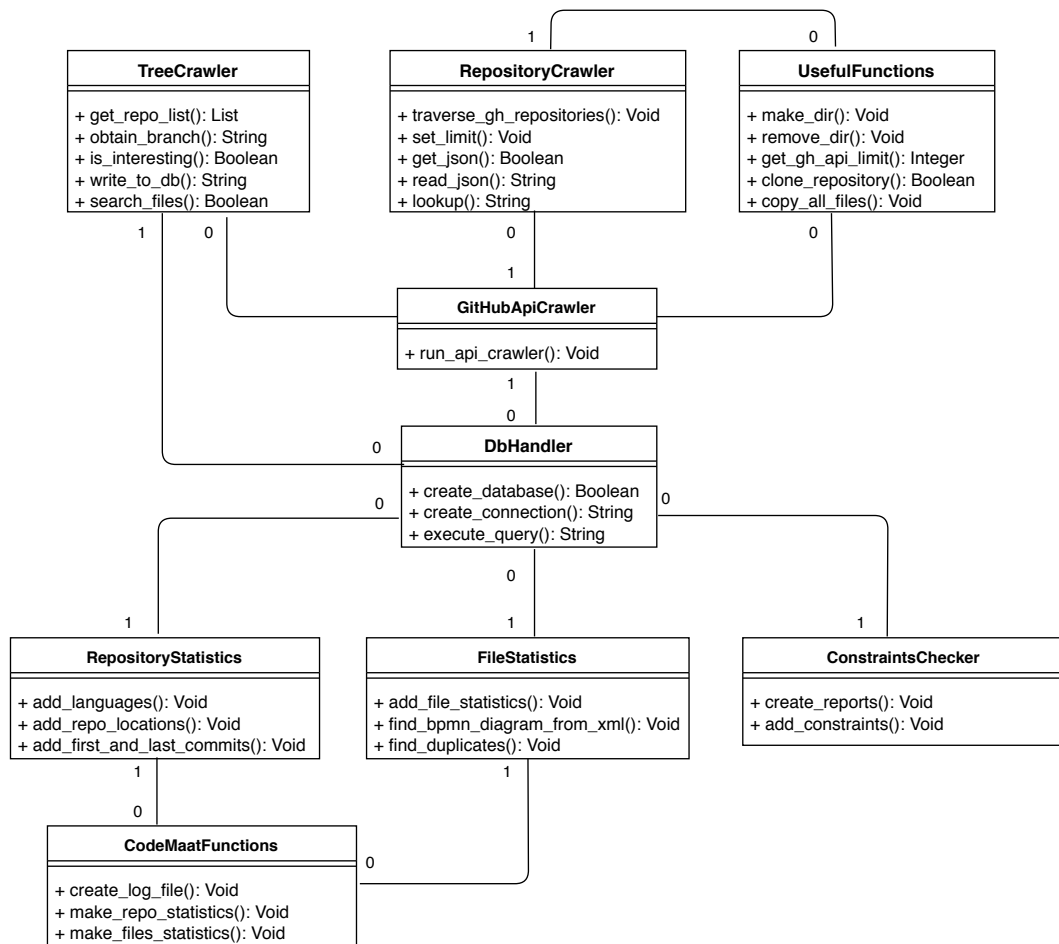


Abbildung 3.4: Klassendiagramm von „GitHubBpmnCrawler“

to_query_projects Datenbanktabelle angeben, zwischen denen alle Repositories nach BPMN-Diagramme zu durchsuchen sind. Das wird gemacht, damit jede Instanz des „GitHubBpmnCrawler“ eigene Menge von Repositories durchsucht. Dann wird eine Liste der Repositoriesnamen und Logins aus to_query_projects Datenbanktabelle ermittelt, die noch nicht durchsucht wurden. Für eine bestimmte Teilmenge aus dieser Liste werden GitHub Repositories nach BPMN-Diagrammen durchsucht. Danach wird das status Attribut in der to_query_projects auf 1 gesetzt, was bedeutet das diese Repositories schon durchsucht wurden. Schließlich wird dieser Vorgang für die nächste Teilmenge der Liste wiederholt. run_api_crawler() terminiert, wenn die Liste komplett bearbeitet wurde. Für die Erledigung der Einzelaufgaben benutzt run_api_crawler() Methoden aus der **RepositoryCrawler**, **TreeCrawler** und **UsefulFunctions** Klassen.

Die **RepositoryCrawler** Klasse hat die zentrale Bedeutung im Programm. Wie im Abschnitt 3.1.3 beschrieben wurde, definiert GitHub API 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer. Um diese Anzahl nicht zu übersteigen, prüft die Methode `set_limit()` die Anzahl der verfügbaren GitHub API Anfragen und wenn sie verbraucht wurden, pausiert das Programm, bis neue 5000 Anfragen zur Verfügung gestellt werden. Andere Methoden dieser Klasse (`traverse_gh_repositories()`, `get_json()`, `read_json()`, `lookup()`) erledigen im Abschnitt 3.1.3.2 bereits beschriebenen Vorgang. Das heißt, für jedes GitHub Repository wird sein

Ordner- und Dateistruktur mithilfe der API http-Anfragen ermittelt und in einer JSON-Datei gespeichert.

Die **TreeCrawler** Klasse enthält Methoden, die es möglich machen, potentielle BPMN-Diagramme innerhalb jeweiligem Repository zu finden. Die Methode *get_repo_list()* ermittelt aus einem Ordner mit heruntergeladenen JSON-Dateien die Namen und Inhaber von Repositorys und liefert als liste zurück. Dann für jedes Element aus dieser Liste wird *search_files()* aufgerufen. In dieser Methode wird die Baumstruktur des entsprechendem Repository traversiert und für jeden Dateinamen die *is_interesting()* Methode aufgerufen. Die Letzte überprüft, ob sich im als Parameter übergebenen Dateinamen einschließlich der Dateiendung das gesuchte Schlüsselwort **bpmn** befindet und liefert im Erfolgsfall *Wahr* oder sonst *Falsch* zurück. Falls das Schlüsselwort **bpmn** ein Teil des Dateinamens war, wird Methode *obtain_branch()* aufgerufen, um *branch* für jeweiliges Repository zu ermitteln. Dies wird benötigt, um das Weblink fürs gefundene potentielle BPMN-Diagramm zu bilden. Nachdem das passiert, wird Methode *write_to_db()* aufgerufen. In letzter wird eine Einfüge-Anfrage mit Repositoryname, Login und ermitteltem Weblink an die *result_files* Datenbanktabelle gesendet.

Die Methoden der **UsefulFunctions** Klasse dienen solchen Zwecken, wie Erzeugen und Löschen eins Ordners oder Kopieren von Dateien. Sie werden aus anderen Klassen aufgerufen.

Die Klasse **DbHandler** spielt die Rolle des Vermittlers zwischen dem Programm und der Datenbank. Sie enthält Methoden für Erzeugung einer Datenbank: *create_database()*, für Erstellung der Verbindung zu einer Datenbank: *create_connection()* und Methode für Ausführung der Datenbankanfragen: *execute_query()*.

In der *execute_query()* Methode wurde der Fall berücksichtigt, wenn mehrere Instanzen des Programms gleichzeitig in dieselbe Tabelle schreiben versuchen. In diesem Fall wird nur ein Schreibvorgang zu selber Zeit durchgeführt. Das bedeutet, dass der Schreibvorgang einer Programminstanz ohne Verzögerungen durchgeführt wird. Bei anderen Programminstanzen, die auch schreiben versuchen, wird die Meldung „database is locked“ (deutsch: Datenbank ist blockiert) ausgegeben. Wenn das passiert, wird das betroffene Programm für 0.3 sec schlafen gelegt und danach wird versucht wieder zu schreiben. Das wird wiederholt, bis der Schreibvorgang erfolgreich durchgeführt wird.

4 Analyse gefundener BPMN-Prozesse

5 Zusammenfassung und Ausblick

Literaturverzeichnis

- [1] Business process modeling notation (bpmn). https://www.service-architecture.com/articles/web-services/business_process_modeling_notation_bpmn.html. Accessed: 2019-04-15.
- [2] formal/13-12-09. <https://www.omg.org/spec/BPMN/2.0.2>. Accessed: 2019-04-15.
- [3] Ghtorrent on the google cloud. <http://ghtorrent.org/gcloud.html>. Accessed: 2019-04-19.
- [4] The ghtorrent project. <http://ghtorrent.org/>. Accessed: 2019-04-19.
- [5] Github facts. <https://github.com/about/facts>. Accessed: 2019-04-19.
- [6] Google bigquery. <https://cloud.google.com/bigquery/>. Accessed: 2019-04-19.
- [7] Rest api v3. <https://developer.github.com/v3/>. Accessed: 2019-04-22.
- [8] The state of the octoverse. <https://octoverse.github.com/>. Accessed: 2019-04-19.
- [9] Thomas Allweyer. *BPMN 2.0 - Business Process Model and Notation - Einführung in den Standard für die Geschäftsprozessmodellierung*. BoD â Books on Demand, Norderstedt, 1. aufl. edition, 2015.
- [10] Kocian Claudia. *Geschäftsprozessmodellierung mit BPMN 2.0 Business Process Model and Notation im Methodenvergleich*. HNU Working Paper Nr. 16. Neu-Ulm University of Applied Sciences, Wileystraße 1, D-89231 Neu-Ulm, 2011.
- [11] Rasmus Voss Jakub Narebski Ferdinando Santacroce, Aske Olsson. *Git: Mastering Version Control*. Packt Publishing Ltd, Birmingham, 2016.
- [12] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pages 384–387, New York, NY, USA, 2014. ACM.
- [13] A. E. Hassan. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, pages 48–57, Sep. 2008.

- [14] Regina Hebig, Truong Ho Quang, Michel R. V. Chaudron, Gregorio Robles, and Miguel Angel Fernandez. The quest for open source projects that use uml: Mining github. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16*, pages 173–183, New York, NY, USA, 2016. ACM.
- [15] K. S. Herzig and A. Zeller. Mining your own evidence. In *Making Software - What Really Works, and Why We Believe It*, pages 517–529. O'Reilly Media, Inc., Sebastopol, 2011.
- [16] Vivek Kale. *Enterprise Process Management Systems - Engineering Process-Centric Enterprise Systems using BPMN 2.0*. CRC Press, Boca Raton, Fla, 2018.
- [17] Gregorio Robles Miguel Angel Fernandez. Extracting software development information from floss projects in github. Paper at SATToSE 2017, http://sattose.wdfiles.com/local--files/2017:schedule/SATToSE_2017_paper_6.pdf, 2017. Accessed: 2019-04-20.
- [18] OMG. *Business Process Model and Notation (BPMN)*. Needham USA(2013), 2013.
- [19] Gregorio Robles. 2016-uml-miner. <https://github.com/LibreSoftTeam/2016-uml-miner>. Accessed: 2019-04-23.
- [20] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel Chaudron, and Miguel Fernández. An extensive dataset of uml models in github. pages 519–522, 05 2017.
- [21] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel R. V. Chaudron, and Miguel Angel Fernandez. An extensive dataset of uml models in github. In *Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17*, pages 519–522, Piscataway, NJ, USA, 2017. IEEE Press.